drv-parts

Module system for packages

Outline

- Facts
- Features
- Why nixos modules
- NixOS ↔ drv-parts
- What is a package ?
- Drv-parts ↔ dream2nix
- Demo
- Problems
- Questions ?

Facts

- Github: github.com/DavHau/drv-parts
- Part of the dream2nix project
- Funded by NLNet
- Started at the last oceansprint.org

Features of drv-parts

- Define packages
- Type checking
- Name checking
- Discoverability
- Documentation
- Separate: Env, flags, options
- Composition

Why the nixos-module system ?

- It exists
- Supported by a large community
- Already has most of the features needed:
  - type/name checking
  - Composability
  - Discoverability
  - (Documentation Generation)

NixOS ↔ drv-parts

	NixOS	**drv-parts**
Ships eval logic	yes	no
Evaluated via	nixpkgs.lib.evalModules	nixpkgs.lib.evalModules
Result attribute	config.system	config.public
Result type	Linux distro	“Package”

What is a package? (config.public)

-> Attrset

Fields (as proposed by @roberth via nix #6507):

- name
- version
- meta
- outputs
- tests
- ${output} for each output

Fields required by Nix CLI:

- drvPath
- outPath
- outputName
- type

drv-parts ↔ dream2nix

drv-parts: collection of low-level package modules

- docs
- env
- binary flags (eg. fooSupport, enableBar, withBaz)
- builtins.derivation
- mkDerivation
- public (package interface)

dream2nix: collection of high-level package modules:

- lock
- python
- nodejs
- pip (requirements.txt, setup.py, poetry.lock, etc.)

Demo Time

Problems

mkDerivation related issues:

- Setup.sh does too much
- Setup hooks don’t compose
- Lists are used a lot

How to manage phases?

- setup.sh
- Setup hooks
- mkDerivation phases
- Systemd style

Debugging:

- Infinite recursions

Dependency management:

- deps / depsModules

Questions ?